

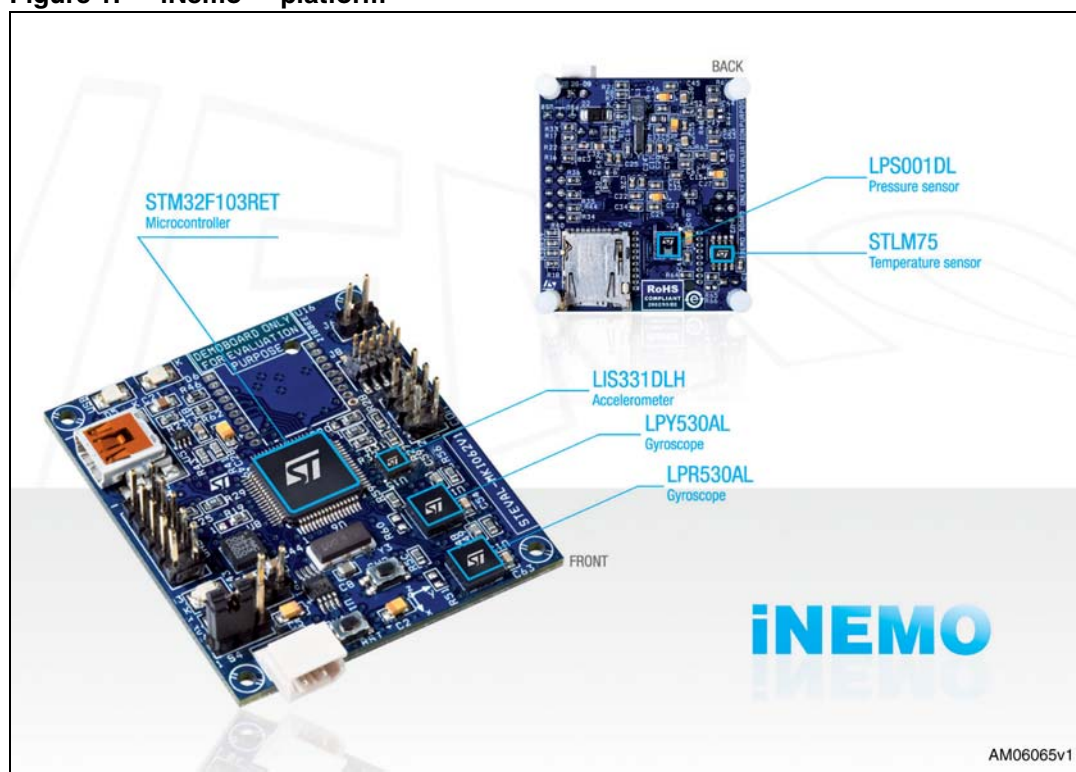
Wrapper in Visual Studio for the STEVAL-MKI062V1

Introduction

This application note describes how to interface a Microsoft® Visual Studio® project with the dynamic link library (DLL) of the application programming interface (API) of the iNEMO™ software development kit (SDK).

The DLL object handles the hardware communication interfacing and is an easy way to achieve (soft) real-time performance. The use of DLLs helps with the modularization of code, code reuse, efficient memory use, and reduction of disk space. Typically, this is preferred when you want to access the iNEMO™ demonstration board (STEVAL-MKI062V1) capabilities directly from a software project without accessing the microcontroller firmware.

Figure 1. iNemo™ platform



Contents

1	Getting started	3
1.1	Power-up	3
1.2	Notes on Microsoft Visual Studio version	3
1.3	iNEMO connection	4
1.4	Releasing the application	5
2	Application code	6
2.1	Package description	6
2.2	Dynamic link library loading	6
2.2.1	Loading the iNEMO_SDK.DLL	6
2.2.2	How to use the DLL functions	7
2.3	iNEMO functions in detail	8
2.3.1	Connecting the PC to iNEMO	8
2.3.2	Disconnecting the connected iNEMO device	9
2.3.3	Checks the iNEMO device connection state	9
2.3.4	Gets the firmware version	9
2.3.5	Starting data acquisition	10
2.3.6	Stopping the current acquisition	10
2.3.7	Getting a data sample	11
2.3.8	Getting the internal buffer usage of the FIFO	12
3	References and related materials	13
4	Revision history	14

1 Getting started

1.1 Power-up

The STEVAL-MKI062V1 board can be supplied by USB. Before connecting the board to a USB port, configure switch S4 as described in the user manual for the board (UM0752).

The STEVAL-MKI062V1 PC software must be pre-installed on the computer before using the C++ code described in this application note, and it is necessary to download iNEMO application 1.0.8 or higher.

1.2 Notes on Microsoft Visual Studio version

The software code described in this application note is compatible with the following versions of the development environment:

- Microsoft Visual Studio 2005
- Microsoft Visual Studio .NET 2003
- Microsoft Visual Studio .NET 2002

As a guideline, users can follow these steps to make a project capable of managing the iNEMO DLL from native C++ code:

1. Start Visual Studio.
2. In the File menu, click New, and then Project.
3. If you are using Visual C++[®] .NET 2002 or 2003, click Visual C++ Projects under Project Types, and then click Win32 Console Project or Win32 Project under Templates. If you are using Visual C++ 2005, click Visual C++ under Project Types, and then click Win32 Project under Templates.
4. In the Name text box, type iNEMO_Cpp_Wrapper, and then click OK.
5. If you are using Visual C++ .NET 2003 or Visual C++ 2005, click Finish. If you are using Visual C++ .NET 2002, click Application Settings, click Console Application, and then click Finish.
6. Open the iNEMO_Cpp_Wrapper file in Code view.

If you receive the error message “Run-Time Check Failure #0”, the value of ESP was not properly saved across a function call. This is usually a result of calling a function declared with one calling convention with a function pointer declared using a different calling convention. When you run the debug build of your application, you can resolve this by the following steps:

1. In Solution Explorer, right-click on your ProjectName, and then click Properties. The ProjectName Property Pages dialog box appears.
2. In the left pane of the ProjectName Property Pages dialog box, click Configuration Properties.
3. Under Configuration Properties, click C/C++, and then click Code Generation.
4. Set the Basic Runtime Checks property to Default.
5. In the ProjectName Property Pages dialog box, click OK.
6. In the File menu, click Save All to save all of the files.

1.3 iNEMO connection

Once the iNEMO demonstration board is connected to a USB port, it is identified by the PC through the virtual COM port driver, and the PC assigns a port number to the iNEMO board.

To identify which port number has been assigned by the PC, follow these steps:

1. On the PC desktop, right click on the My Computer icon and click on Manage (see [Figure 2](#)).
2. In Computer Management under System Tools, click on Device Manager to see all the devices connected to the PC in the right-hand pane.
3. In the right-hand pane, click the “+” next to Ports (COM & LPT) to see the list of available ports. In this list you should find the STEVAL-MKI062V1 iNEMO Platform (COMx) string (see [Figure 3](#)).

The string containing the COM port number assigned to the board is used by the source code described in [Section 2.3.1: Connecting the PC to iNEMO](#).

Note: *The COM port number assigned is valid for one session only; when the board is unplugged and plugged in again (especially into a new USB port) the COM port number assigned could be different. It should therefore be checked again using the procedure described above.*

Figure 2. Identifying the COM port used by the iNEMO board - first step

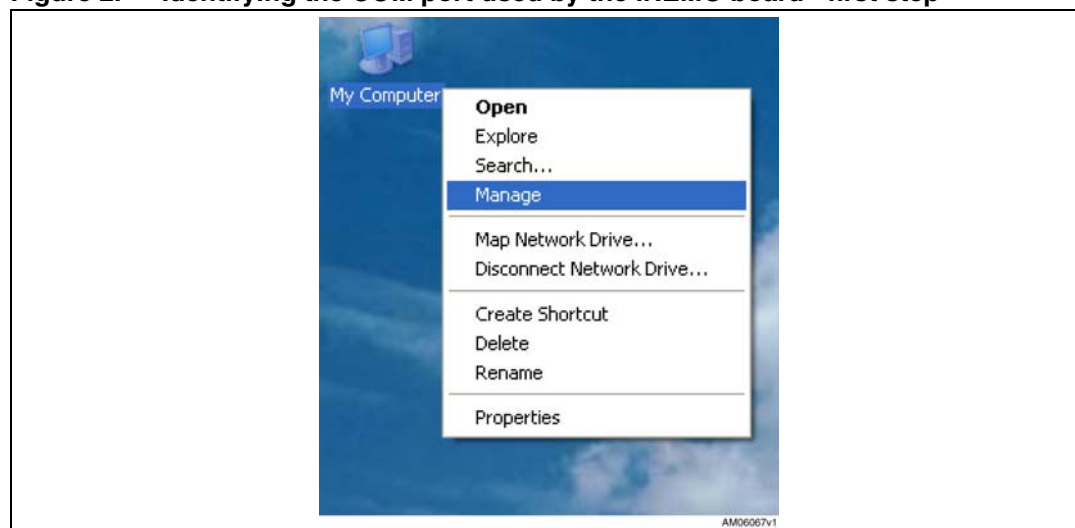
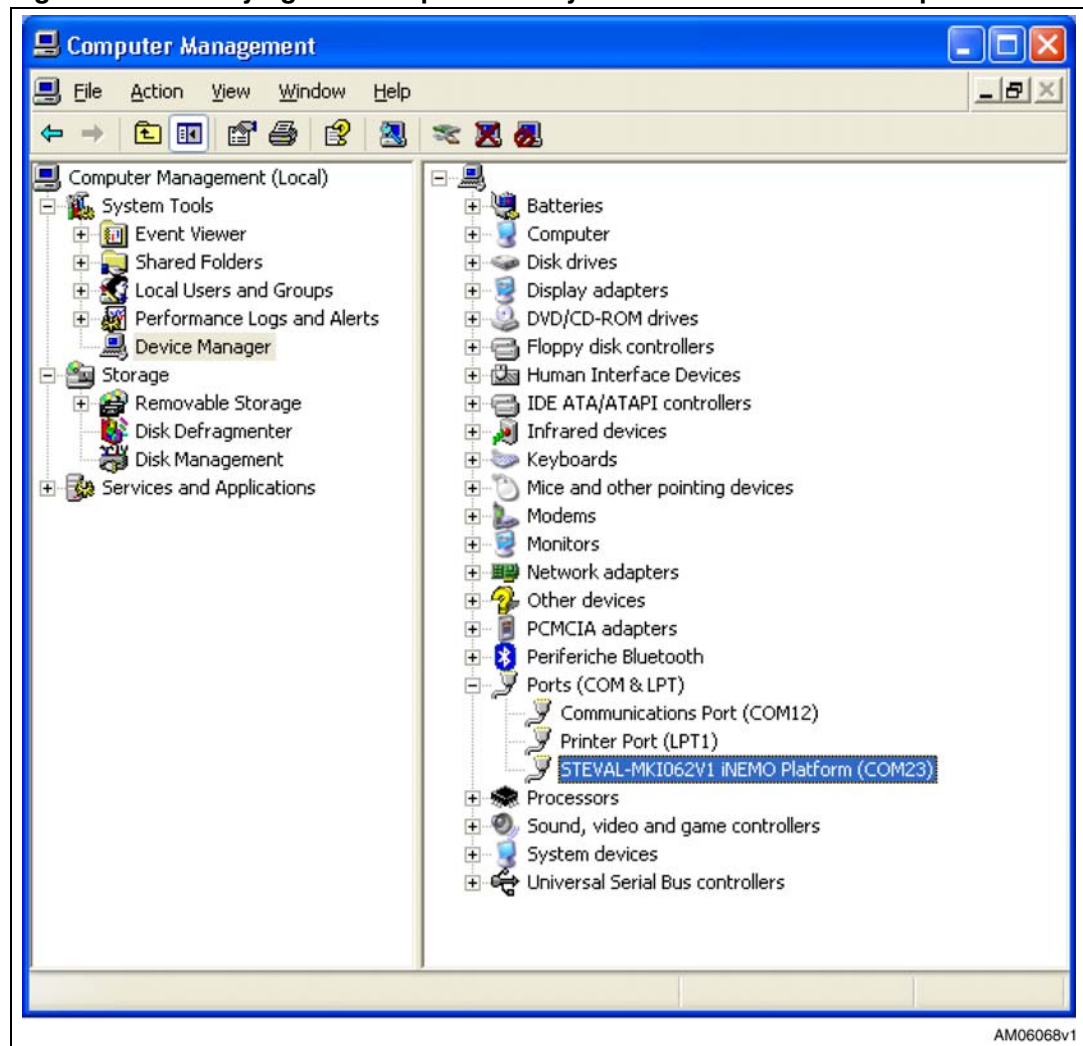


Figure 3. Identifying the COM port used by iNEMO board - second step

1.4 Releasing the application

You can freely release your own application which is linked with iNEMO DLL. Because the iNEMO_SDK.dll is dependent on the PL_001.dll, both DLLs must be located in the same folder. It is recommended to have the two DLLs in the application folder to avoid conflicts with different versions.

2 Application code

2.1 Package description

The example code provided below uses the files contained in the subfolder:

\STMicroelectronics\iNEMO Application\Redist

(the exact folder name may be different on your computer)

This subfolder contains:

- iNEMO_SDK.chm - Help file for the SDK
- iNEMO_SDK.dll - Dynamic link library of the SDK
- iNEMO_SDL.h - Header file for the iNEMO SDK dll
- iNEMO_SDK_Wrapper.cs - Wrapper file in C#
- PL_001.dll - PL_001 Dynamic Link Library

The source code in the sections that follow can be used as a guideline to develop your own application.

2.2 Dynamic link library loading

Generally, two loading methods allow you to call the exported DLL functions when loading a DLL in your application. The two loading methods are *static* load and *dynamic* load.

In static loading, the loading/unloading of the DLL is managed automatically by the run-time framework when the user runs/closes the application. In this case, in the Visual Studio C++ application you need to include the iNEMO_SDK.h and the relative import library file (iNEMO_SDK.lib).

In dynamic loading, the application explicitly loads/unloads the DLL through the LoadLibrary and FreeLibrary functions.

The DLL of the application programming interface (API) of the iNEMO software development kit (SDK) can be easily linked to your application in dynamic mode. In this way your application calls either the LoadLibrary function or the LoadLibraryEx function to load the DLL at run-time. After the DLL is successfully loaded, you can use the GetProcAddress function to obtain the address of the exported DLL function that you want to call. When you load the DLL dynamically, you do not need an import library file.

Dynamic loading offers two advantages: the startup performance of the application, and the capability of the application to branch and load different modules if required (i.e. load two versions of the same DLL).

2.2.1 Loading the iNEMO_SDK.DLL

The following code is an example of a Win32 Application project that calls the exported DLL function in the iNEMO_SDK.dll file using dynamic loading:

```
// iNEMO_Cpp_Wrapper.cpp  
//  
...
```

```

HINSTANCE hinstDLL;
BOOL fFreeDLL;

...

hinstDLL = LoadLibrary("iNEMO_SDK.dll");
if (hinstDLL != NULL)
{
    // Manage the functions here
    ...

    fFreeDLL = FreeLibrary(hinstDLL);
}

...

```

When you compile and link the iNEMO_Cpp_Wrapper application, the Windows® operating system searches for the iNEMO_SDK DLL in the following locations, in the following order:

1. Application folder
2. The current folder
3. Windows system folder
4. Windows folder

The FreeLibrary instruction frees the loaded dynamic-link library (DLL) module, the module is unloaded from the address space of the calling process, and the handle is no longer valid. If the function succeeds, the fFreeDLL return value is non-zero. Otherwise, if the function fails, the return value is zero. To obtain extended error information, call the GetLastError function.

- Note:**
- 1 To return the iNEMO board to its initial state, unplug and plug in the board again.
 - 2 The iNEMO_SDK.DLL is dependent on the PL_001.DLL, therefore this DLL must be in your search path.

2.2.2 How to use the DLL functions

The iNEMO_SDK DLL exports the following functions:

- INEMO_SDK_API INEMO_SDK_Connect(LPCSTR lpszConnectionString);
- INEMO_SDK_API INEMO_SDK_Disconnect();
- INEMO_SDK_API INEMO_SDK_GetBufferUsage(double *pPerc);
- INEMO_SDK_API INEMO_SDK_GetSample(PFrameData_t pFrame);
- INEMO_SDK_API INEMO_SDK_GetVersion(LPSTR lpVersion, int *lpnSize);
- INEMO_SDK_API INEMO_SDK_IsConnected();
- INEMO_SDK_API INEMO_SDK_Start(int nFreq, int nSamples);
- INEMO_SDK_API INEMO_SDK_Stop();

To use these functions in your project, you must define a new type:

...

```
typedef int (WINAPI *DLLPROC) ();  
...
```

And declare a variable to be linked to the functions you wish to use:

```
...  
DLLPROC INEMO_SDK_Function;  
...
```

Thus, the program must use the returned handle in the GetProcAddress function to obtain the address of the function of the DLL:

```
INEMO_SDK_Function = (DLLPROC) GetProcAddress(hinstDLL, "...");  
if (INEMO_SDK_Function != NULL)  
    INEMO_SDK_Function();
```

2.3 iNEMO functions in detail

The following subsections describe the relative software code for each function available within the iNEMO software development kit.

Users can manage errors by adding the variable below. For each function, an error code is returned if an error is present, while a "0" is returned if there is no error.

```
...  
int iNEMO_Return;  
...
```

2.3.1 Connecting the PC to iNEMO

The following code opens the Virtual COM port assigned by the Windows operating system:

```
...  
typedef int (WINAPI *DLLPROC_CON)(LPCSTR);  
...  
DLLPROC_CON INEMO_SDK_ConnectFunction;  
...  
INEMO_SDK_ConnectFunction = (DLLPROC_CON) GetProcAddress(hinstDLL,  
"INEMO_SDK_Connect");  
if (INEMO_SDK_ConnectFunction != NULL)  
    iNEMO_Return = INEMO_SDK_ConnectFunction("PL_001:COM23, BD115200");
```

The developer needs to substitute COM23 with the exact COM number assigned by the PC (see [Section 1.3: iNEMO connection](#) for details).

The variable iNEMO_Return returns a "0" if there is no error, otherwise it returns an error code.

2.3.2 Disconnecting the connected iNEMO device

The following code closes the Virtual COM port channel:

```
...
typedef int (WINAPI *DLLPROC) ();
...
DLLPROC INEMO_SDK_DisconnectFunction;
...
INEMO_SDK_DisconnectFunction = (DLLPROC) GetProcAddress(hinstDLL,
    "INEMO_SDK_Disconnect");
    if (INEMO_SDK_DisconnectFunction != NULL)
        iNEMO_Return = INEMO_SDK_DisconnectFunction();
```

The variable iNEMO_Return returns a "0" if there is no error, otherwise it returns an error code.

2.3.3 Checks the iNEMO device connection state

The following code helps to verify if the connection occurs correctly:

```
...
typedef int (WINAPI *DLLPROC) ();
...
DLLPROC INEMO_SDK_IsConnectFunction;
...
INEMO_SDK_IsConnectFunction = (DLLPROC) GetProcAddress(hinstDLL,
    "INEMO_SDK_IsConnected");
    if (INEMO_SDK_IsConnectFunction != NULL)
        iNEMO_Return = INEMO_SDK_IsConnectFunction();
```

The variable iNEMO_Return returns a "0" if iNEMO is not connected, otherwise it is successfully connected.

2.3.4 Gets the firmware version

The user can retrieve the firmware version by using the following code:

```
...
typedef int (WINAPI *DLLPROC_GFWV)(LPSTR, int*);
...
DLLPROC_GFWV INEMO_SDK_GetFWVersionFunction;
...
char sVersion[] = "iNEMO String for Firmware Version";
```

```

int iSize = 30;
...

INEMO_SDK_GetFWVersionFunction = (DLLPROC_GFWV)
GetProcAddress(hinstDLL,
               " INEMO_SDK_GetVersion");
if (INEMO_SDK_GetFWVersionFunction != NULL)
    iNEMO_Return = INEMO_SDK_GetFWVersionFunction(sVersion,
&iSize);

```

The variable iNEMO_Return returns a "0" if there is no error, otherwise it returns an error code.

2.3.5 Starting data acquisition

The following code starts the data acquisition of the iNEMO board:

```

...
typedef int (WINAPI *DLLPROC_START)(int, int);
...
DLLPROC_START INEMO_SDK_StartFunction;
...
int nFreq = 1;
int nSample = 5;
...
INEMO_SDK_StartFunction = (DLLPROC_START) GetProcAddress(hinstDLL,
                  "INEMO_SDK_Start");
if (INEMO_SDK_StartFunction != NULL)
    iNEMO_Return = INEMO_SDK_StartFunction(nFreq, nSample);

```

The variable iNEMO_Return returns a "0" if there is no error, otherwise it returns an error code. Moreover, nFreq is the acquisition frequency (Hz) and nSamples is the number of samples to receive. If the latter is "0", the iNEMO board sends data in infinite loop mode until an INEMO_SDK_Stop command is received. Otherwise the board sends only the finite number of samples indicated in nSamples parameter.

The nFreq parameter accepts the values 1, 10, 25 or 50, while the nSamples parameter accepts values in the range of 0 - 65535. As described above, this parameter is also used to choose the acquisition mode.

2.3.6 Stopping the current acquisition

The following code stops the acquisition phase:

```

...
typedef int (WINAPI *DLLPROC) ();

```

```

...
DLLPROC INEMO_SDK_StopFunction;
...
INEMO_SDK_StopFunction = (DLLPROC) GetProcAddress(hinstDLL,
"INEMO_SDK_Stop");
    if (INEMO_SDK_StopFunction != NULL)
        iNEMO_Return = INEMO_SDK_StopFunction();

```

The variable iNEMO_Return returns a "0" if there is no error, otherwise it returns an error code.

2.3.7 Getting a data sample

As described above, the INEMO_SDK_Start instruction starts an internal process of the Dynamic Link Library which collects the data from the iNEMO board in a FIFO (first in, first out) list. To obtain the data from this list, the following instructions must be used:

```

...
typedef int (WINAPI *DLLPROC_GS)(PFrameData_t);
...
DLLPROC_GS INEMO_SDK_GetSampleFunction;
FrameData_t frame;
...
for (int i = 0; i < nSample; ++i)
{
    INEMO_SDK_GetSampleFunction = (DLLPROC_GS)
GetProcAddress(hinstDLL, "INEMO_SDK_GetSample");
    if (INEMO_SDK_GetSampleFunction != NULL)
        iNEMO_Return = INEMO_SDK_GetSampleFunction(&frame);

    // Manage here the frame variable
}

```

where frame is a C-language structure organized as follows:

```

typedef struct
{
    SampleTimeStamp // timestamp (milliseconds from start)
    Accelerometer   // Accelerometer component (X, Y, Z) in mg unit
    Gyroscope        // Gyroscope component (X, Y1, Y2, Z) in dps unit
    Magnetometer     // Magnetometer component (X, Y, Z) in mGa unit
    Pressure         // Pressure value in 1/10 of mbar unit
}

```

```

    Temperature      // Temperature value in 1/10 of °C unit
}

```

Note: *The FIFO filling is dictated by the frequency indicated with the `nFreq` parameter, since the use of the `INEMO_SDK_GetSample` instruction must correspond with this frequency. Otherwise, a faster `INEMO_SDK_GetSample` returns an empty frame structure and an excessively slow `INEMO_SDK_GetSample` returns values which are too old.*

To verify the effectiveness of the frame structure, it is suggested to check the returned `INEMO_Return` value, which could be:

`INEMO_SDK_ERROR_NONE` = No error

`INEMO_SDK_ERROR_STACK_EMPTY` = The stack of data is empty

2.3.8 Getting the internal buffer usage of the FIFO

The following code checks the internal buffer usage:

```

...
typedef int (WINAPI *DLLPROC_GBU)(double*);
...
DLLPROC_GBU INEMO_SDK_GetBufferUsageFunction;
double perc = 0;
...
INEMO_SDK_GetInternalBufferFunction = (DLLPROC_GBU)
GetProcAddress(hinstDLL, "INEMO_SDK_GetBufferUsage");
if (INEMO_SDK_GetBufferUsageFunction != NULL)
    iNEMO_Return = INEMO_SDK_GetBufferUsageFunction(&perc);

```

The variable `iNEMO_Return` returns a "0" if there is no error, otherwise it returns an error code. Meanwhile, the variable `perc` contains the actual percentage of internal buffer FIFO usage.

3 References and related materials

The following additional information for the STEVAL-MKI062V1 is available at www.st.com/inemo:

- STEVAL-MKI062V1 data brief
- User manual UM0752: STEVAL-MKI062V1, iNEMO™ (iNErtial MOdule) demonstration board based on MEMS devices and STM32F103Rx
- Application Note AN3138: Wrapper in MATLAB for the STEVAL-MKI062V1

4 Revision history

Table 1. Document revision history

Date	Revision	Changes
08-Jul-2010	1	Initial release.

Please Read Carefully:

Information in this document is provided solely in connection with ST products. STMicroelectronics NV and its subsidiaries ("ST") reserve the right to make changes, corrections, modifications or improvements, to this document, and the products and services described herein at any time, without notice.

All ST products are sold pursuant to ST's terms and conditions of sale.

Purchasers are solely responsible for the choice, selection and use of the ST products and services described herein, and ST assumes no liability whatsoever relating to the choice, selection or use of the ST products and services described herein.

No license, express or implied, by estoppel or otherwise, to any intellectual property rights is granted under this document. If any part of this document refers to any third party products or services it shall not be deemed a license grant by ST for the use of such third party products or services, or any intellectual property contained therein or considered as a warranty covering the use in any manner whatsoever of such third party products or services or any intellectual property contained therein.

UNLESS OTHERWISE SET FORTH IN ST'S TERMS AND CONDITIONS OF SALE ST DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY WITH RESPECT TO THE USE AND/OR SALE OF ST PRODUCTS INCLUDING WITHOUT LIMITATION IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE (AND THEIR EQUIVALENTS UNDER THE LAWS OF ANY JURISDICTION), OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

UNLESS EXPRESSLY APPROVED IN WRITING BY AN AUTHORIZED ST REPRESENTATIVE, ST PRODUCTS ARE NOT RECOMMENDED, AUTHORIZED OR WARRANTED FOR USE IN MILITARY, AIR CRAFT, SPACE, LIFE SAVING, OR LIFE SUSTAINING APPLICATIONS, NOR IN PRODUCTS OR SYSTEMS WHERE FAILURE OR MALFUNCTION MAY RESULT IN PERSONAL INJURY, DEATH, OR SEVERE PROPERTY OR ENVIRONMENTAL DAMAGE. ST PRODUCTS WHICH ARE NOT SPECIFIED AS "AUTOMOTIVE GRADE" MAY ONLY BE USED IN AUTOMOTIVE APPLICATIONS AT USER'S OWN RISK.

Resale of ST products with provisions different from the statements and/or technical features set forth in this document shall immediately void any warranty granted by ST for the ST product or service described herein and shall not create or extend in any manner whatsoever, any liability of ST.

ST and the ST logo are trademarks or registered trademarks of ST in various countries.

Information in this document supersedes and replaces all information previously supplied.

The ST logo is a registered trademark of STMicroelectronics. All other names are the property of their respective owners.

© 2010 STMicroelectronics - All rights reserved

STMicroelectronics group of companies

Australia - Belgium - Brazil - Canada - China - Czech Republic - Finland - France - Germany - Hong Kong - India - Israel - Italy - Japan - Malaysia - Malta - Morocco - Philippines - Singapore - Spain - Sweden - Switzerland - United Kingdom - United States of America

www.st.com

